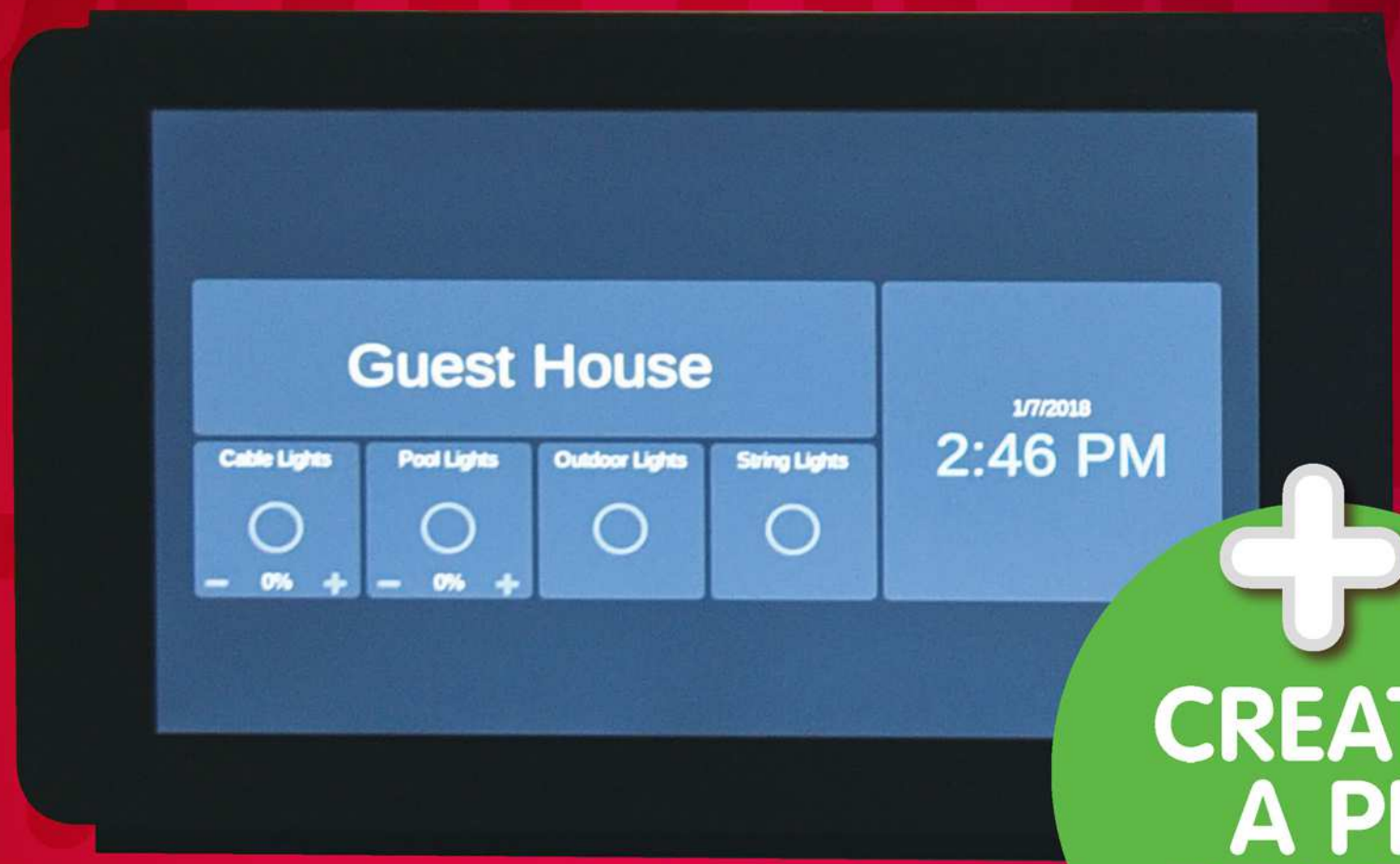
# RasPi

DESIGN
BUILD
CODE

45

Get hands-on with your Raspberry Pi

## CONTROL YOUR HOUSE FROM YOUR PI




+ CREATE A PI SERVER

MAKE A HOME

# AUTOMATION PANEL

**Plus** Find your phone with a Raspi

# Welcome

Another issue and another chance to demonstrate the powerful versatility of the Raspberry Pi. This month we show you how to create a network for Raspberry Pis all connected to a central server, using a great tool called Piserver. We also demonstrate an elegant home-automation interface devised and built by a Pi enthusiast in San Francisco and reveal an ingenious way of finding and tracking phones using a Raspberry Pi 3. Other highlights include accessing the Tor network and managing huge amounts of data using a full Relational DataBase Management System. Is there anything they can't do?

## Get inspired

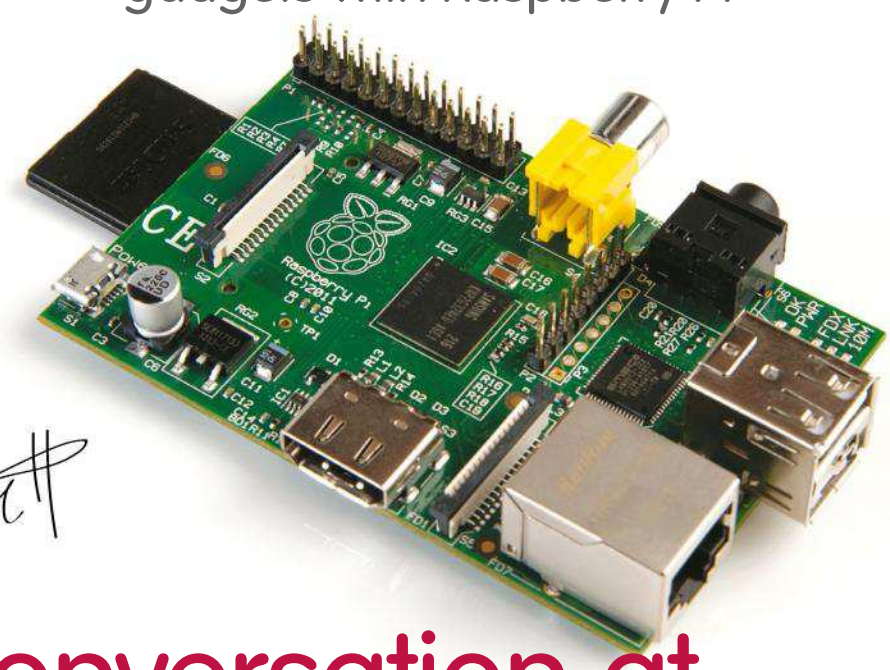
Discover the RasPi community's best projects

## Expert advice

Got a question? Get in touch and we'll give you a hand

## Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi



Editor

From the makers of

LinuxUser & Developer

## Join the conversation at...

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com


# Contents

# Create a Raspberry Pi network with PiServer Tool

## Use PiServer to easily set up a network of Pis connected to a central server, which you control

THE PROJECT ESSENTIALS

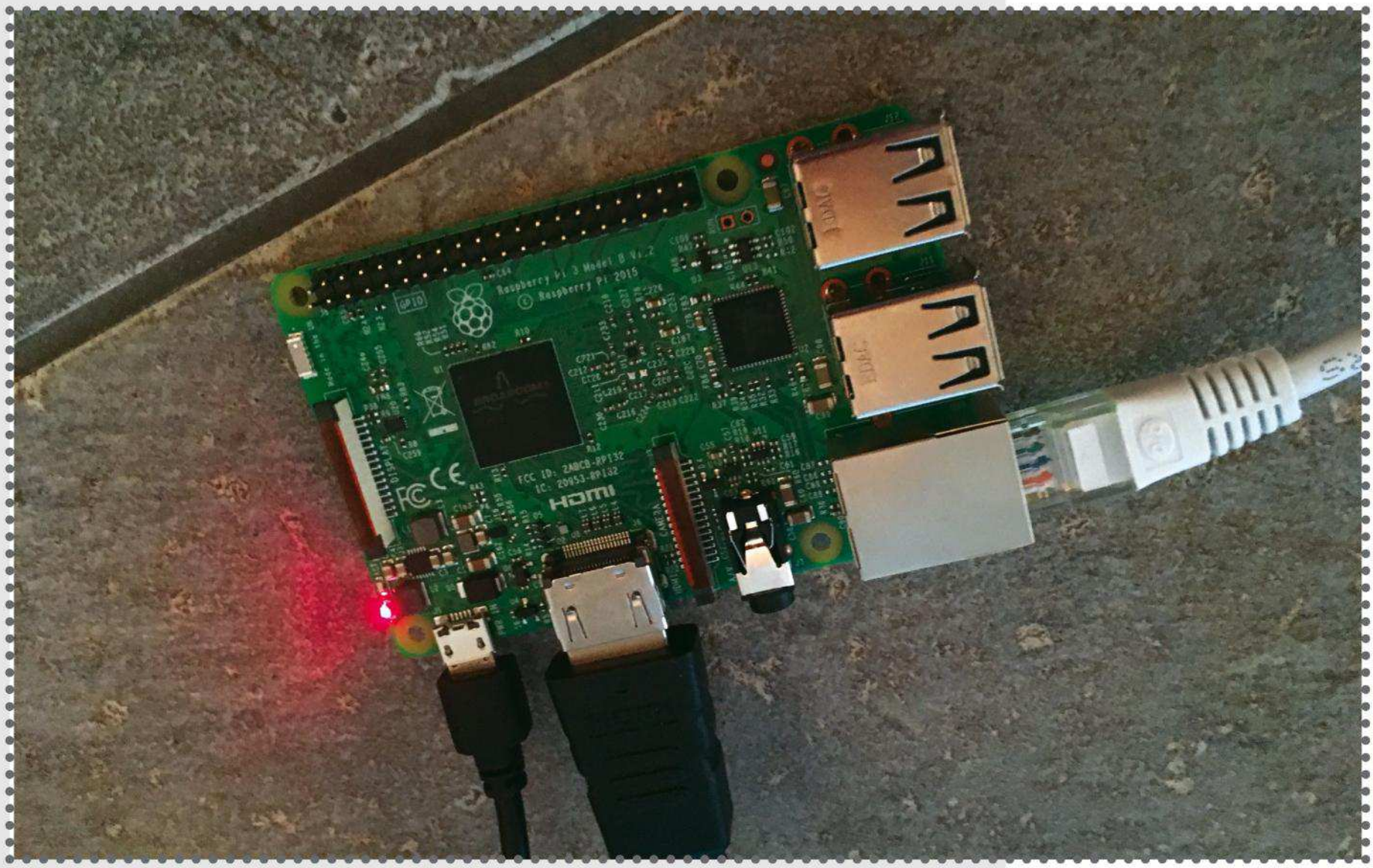
**PiServer**
www.raspberrypi.org/blog/piserver

The good people of the Raspberry Pi Foundation have outdone themselves once again with the release of PiServer, a tool which enables you to easily set up PXE (Pixie) network booting. In plain English, this means you can set up a network of Raspberry Pis, each of which are connected to a single server. The Pis can boot over the network without any need for microSD cards and you, as the server admin, can control user accounts and accessible files.

The fact that every Pi, as a network 'client', uses the same accessible files makes PiServer perfect for classroom environments or workplaces where you may want to use Raspberry Pis for specific purposes such as teaching or operating machinery.

You can create as many users as you need during setup. These users don't have root privileges, meaning they cannot install new software; this makes for a much better and safer learning environment.

In order to get started setting up your network, you'll need at least one Raspberry Pi 3 and an Ethernet cable to connect each one to the router or network hub. The PiServer




tool is currently available via Debian with Raspberry Pi Desktop. Follow the steps in the guide below to install it on a dedicated PC for best results, although you can also set up your server using a virtual machine or even a Raspberry Pi (see below).

## 01 Enable network boot

Each Raspberry Pi you want to use with PiServer must have network boot enabled. To do this you'll need a microSD card with Raspbian or Raspbian Lite preinstalled. To enable network boot, just add the line program_usb_boot_mode=1 to the file config.txt in /boot. You can do this manually on your computer using a card reader, or by opening a terminal on the Pi itself and running the command echo program_usb_boot_mode=1 | sudo tee -a /boot/config.txt. Power off the Pi and remove

"you can also set up your server using a virtual machine"

the microSD once this is done.

## 02 Set up your server

In order to set up a server for your Pis, you'll need either a dedicated x86 computer or a virtual machine onto which you can install Debian with Raspberry Pi Desktop, which includes the PiServer tool. To get started, visit www.raspberrypi.org/downloads/raspberry-pi-desktop and download the ISO file. While you can run the OS in Live mode, we recommend you opt for a 'Graphical Install' to the hard disk. Make sure you have at least 16GB of free space on the drive to contain the Pi's file system and user data.

## 03 Customise the server display

By default, Debian with Raspberry Pi Desktop uses a restrictive 640x480 resolution. To fix this, restart the machine and hold C while it boots to enter the GRUB prompt. Enter the command videoinfo and note which resolutions are supported by your graphics card. Boot to the Debian desktop and open the Terminal. Run sudo nano /etc/default/grub. Remove the # at the start of the line #GRUB_GFXMODE and change it to your desired resolution, for example 1024x768. Press Ctrl+X, followed by Y and then Enter to save and exit. Next, run update-grub and restart to apply your new resolution.

## 04 Register Pis with the server

Connect your Raspberry Pi(s) to your network router or hub via Ethernet and power it on. Return to the Debian desktop and go to Preferences > PiServer. Read through the introduction, then click 'Next'. From this screen you can select the client(s) to add; if you don't see all of your Pis, check that the listed 'network interface' is correct.

Failing this, repeat the first step to enable network boot. If it is enabled, the LEDs both on the Pi and connector should be active. Click 'Next' to continue.


## 05 Create user accounts

You must create at least one user account so that people using your Pis can log in. Remember that this account is created on the server itself and will use the files stored there. This means, for instance, that if you create the user 'Alice' with the password 'password123', she can log in using these credentials on any of the Pi's connected to your server and will see the same files.

You can add and remove user accounts after setup if you want. Fill in the fields, then click 'Next' to continue.

## 06 Install the operating system

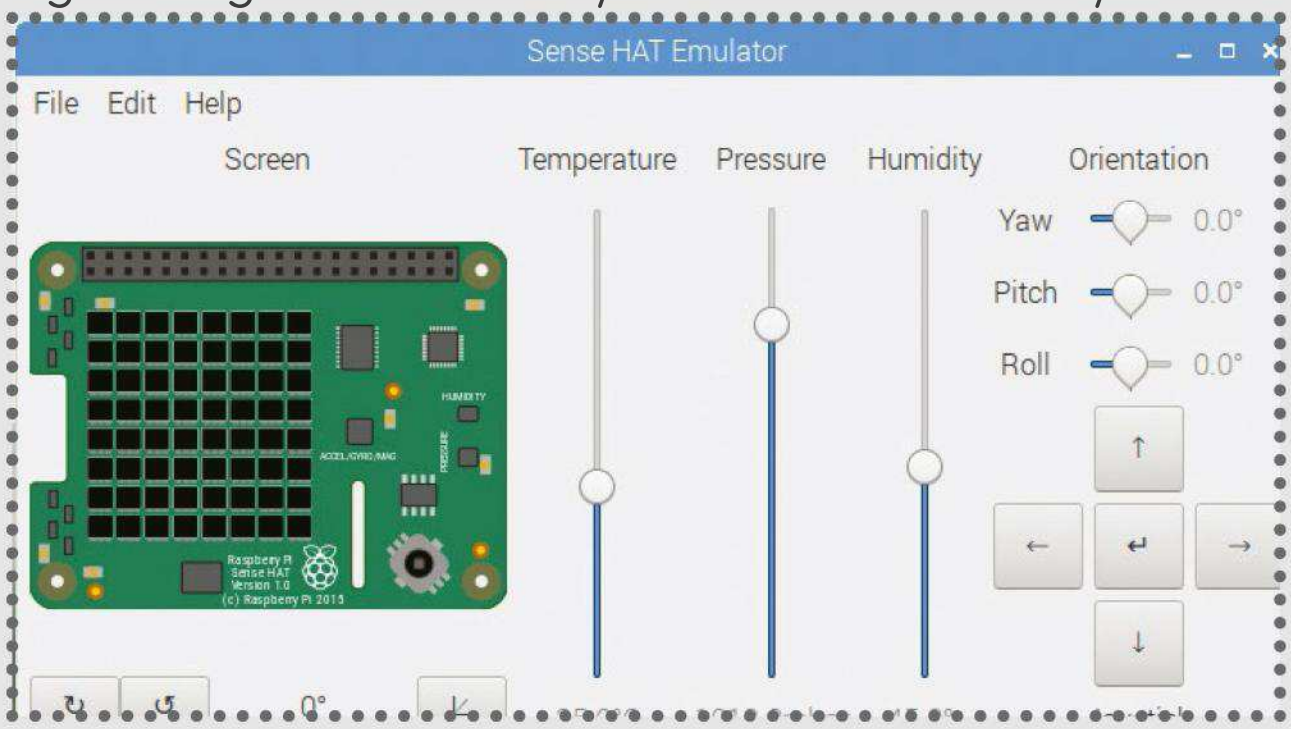
Use the Add Software screen to choose which OS to install. At the time of writing only Raspbian and Raspbian Lite are available, although the Raspberry Team is confident other operating systems will be supported in future. PiServer also supports installing an OS via local files at a URL, although it's not clear if any other Pi-

"PiServer also supports installing an OS via local files at a URL"

compatible systems currently work with it. If you're handy at programming, head to /var/lib/piserver/scripts to see how these operating system images are created. Wait for the software to install before clicking 'Next'.

## 07 Start your Pis

Once installation is complete, restart each Pi connected to your network and make sure that the mouse, monitor and keyboard is connected to each one. Remember that the keyboard must have the same layout as the one connected to your server. Users will need to log in using the credentials you created earlier. They can




use preinstalled software, change the background, mount drives and connect to the internet. Root access is disabled, however, so they cannot make system-wide changes such as installing software.

## 08 Edit users and clients

The PiServer tool enables you to add new users and/or Pis to your network as necessary. To edit user accounts, simply click the Users tab on the left side and click either Add or Remove. As the server admin you can also reset the password for users from here. Use the Clients

tab to add more Pis to your network; the layout here is the same as in Step 4. Use the Edit button to give each Pi a description, for example Classroom Pi 1.

## 09 Install software

As the Pi users don't have root privileges, they can't update the system nor install new programs. Click the Software tab in PiServer to remedy this. From here you can choose to remove or add a new OS altogether. Alternatively click Shell to be taken to the command prompt, where as the root user you can perform operations such as downloading new programs. Use the shell to run apt-get update and apt-get upgrade so that users have the most recent version of Raspbian.

## 10 Change DHCP server settings

In most cases your server is connected to a router, so you'll want it to act as a proxy DHCP server. In other words, if you have a router which assigns IP addresses to devices on the network, your Debian server will not interfere with this in any way while interacting with the Raspberry Pis. However, you can select 'Act as a Standalone DHCP Server' if you want. Check that your desired range of IPs, netmask and gateway addresses are correct and click Save.

"To edit user accounts, simply click the Users tab on the left side"

# Open source smart home touchscreen

A stylish smart home project in San Francisco demonstrates an elegant interface for home automation without proprietary parts.

This project is an elegant solution to a tricky problem with the affordable but useful 7-inch Raspberry Pi touchscreen. Peter Monaco's mission was to create a touchscreen that sat flush to a wall as an interface for home automation without any ugly dangling wires and, as you can see, it has worked beautifully.

**Your bezel frame came out really well. You're obviously experienced with 3D printing – what printer do you use and do you have any advice for others wanting to replicate your project?**

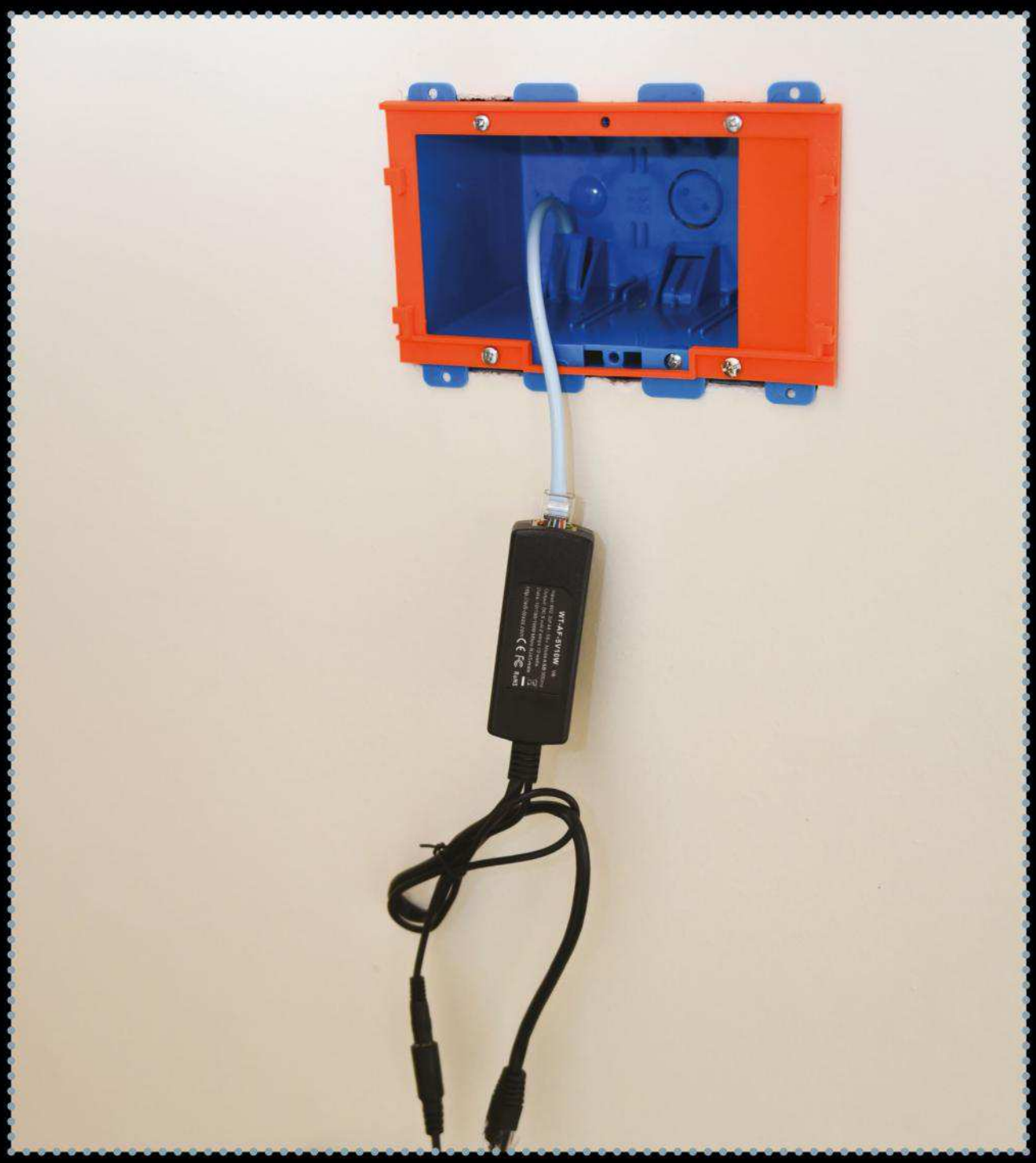
I've been 3D printing for a little over two years. I have a FlashForge Creator Pro, and use SketchUp for all my design work. I enjoy designing little household items that make life more efficient, like clips to hold the Christmas lights to the banister, a caddy to hold my Wi-Fi access point on the wall, or a clip to fix a broken watch band. For this project, I printed all the parts in PLA, which is my favorite material. These were some of the largest parts I've made in PLA, and I did have some problems with them curling at first. I was able to fix the curling by raising the print bed temperature to 50 degrees Celsius. Other settings included a print speed of 70 mm/s and nozzle temperature of 200 degrees Celsius.

**The design has produced a sleek smart home touchscreen. What space-saving techniques did you need to employ?**

This project was an uphill battle all the way. I wanted the screen to be nearly flush with the wall, so the

**Peter Monaco** lives in the San Francisco Bay Area and is a software engineer. He's currently working at the Connectivity Lab at Facebook, which is an innovation team working to develop economically sustainable technologies.

electronics needed to go into an electrical [back] box in the wall. The Pi hanging off the back of the touchscreen barely fit into a 3-gang electrical box, and only after a few modifications (of which more later). After squeezing it into the box, there wasn't much room left for anything else, but I needed to find a space for the high-voltage connections. I designed a set of three walls and some slides that allow them to be inserted and removed easily. These create an L-shaped volume in the rear-right corner of the box. This space is just large enough for some Romex [sheathed cable], a few connectors, and a USB power



**Let** Mounting the panel flush to the wall and powering it was going to be a challenge. Initially, this led to running Romex cable wiring to the electricity box and physically isolating the Romex and the USB transformer from the Pi and touchscreen by 3D-printing a partition for the box. Although it's a clever solution, messing about with high voltage is not for the inexperienced. We'd recommend using the Power over Ethernet method that's pictured above. Subsequently, based on community feedback, this is what Peter has done for his own project. Consider yourself warned.

## THE PROJECT ESSENTIALS

**Raspberry Pi Zero (with enclosure)**

**Biodegradable Arboblend chassis**

**LCD touchscreen**

**Sunpartner transparent solar panel**

**Swappable side panels**

**Wi-Fi antenna**

**4 x AA batteries**

**SD card reader**

**USB power input**

adaptor. Once I found a way to squeeze everything into the box, I designed a faceplate that screws to the box, and a bezel that slides into it. Once it's assembled, the bezel is all that's visible.

**What modifications did you need to make to the Raspberry Pi?**

Like I said, space was at a premium. The Pi wouldn't fit into the electrical box in the vertical dimension without some small modifications. The Pi normally attaches to the touchscreen's adaptor board via some jumper cables, but those cables were hanging too far off the board. I decided to snip them and solder them directly to the adaptor board. This change reduced the vertical dimension of everything by about half an inch, which was all I needed.

Also, if you want to attach an Ethernet cable to the Pi, it has to make a sharp U-turn to fit in the box. The cable I was using had a stiff end on it, so I added a home-made extender out of Cat 5 cable, which is more flexible.

**You chose Home Assistant for the interface. Does this mean you've got plans for other home automation projects in the future?**

I'm just starting to experiment with home automation. I've added a few Wi-Fi light switches (TP-Link HS200) and they're working well. I plan to add some sensors to track energy usage, and possibly some cameras. But it was important to me that my home automation hub be open source – I didn't like the idea of

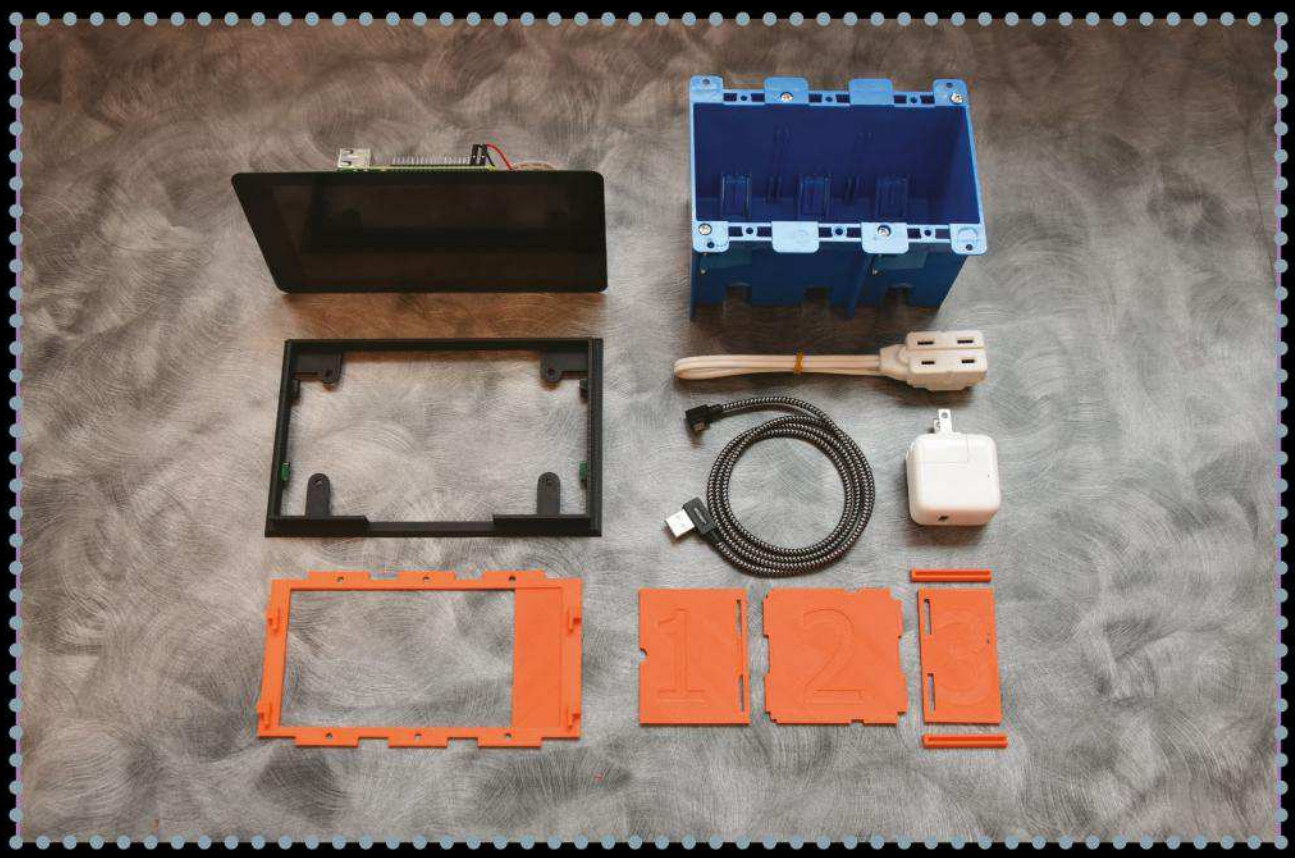
committing to one brand and trusting commercial software to run the home. I looked around and settled on the Home Assistant project (https://home-assistant.io). It has adaptors for hundreds of components, and is totally open source. It was super-easy to set up on a Raspberry Pi running in the closet. Then I found the HADashboard sub-project, which provides a really clean, attractive UI for Home Assistant. That's when I decided to find a way to mount the Raspberry Pi touchscreen in an elegant way.


**Down** After attaching the faceplate to the box with a few electrical box screws, Peter attached a PoE splitter to his Cat 5 cabling, which gave him an Ethernet cable and a Micro-USB port for connecting the Pi for network access and power, respectively. In his case, he found the Ethernet cable from the splitter was too stiff to be turned sharply inside the small box, so he had to make an extension. Once he'd pushed all the wires into the box, connected the 3D-printed bezel to the face place and his Cat 5 cable to his Power over Ethernet source, his touchscreen was connected and ready to go.

**What were the challenges and would you do anything differently now?**

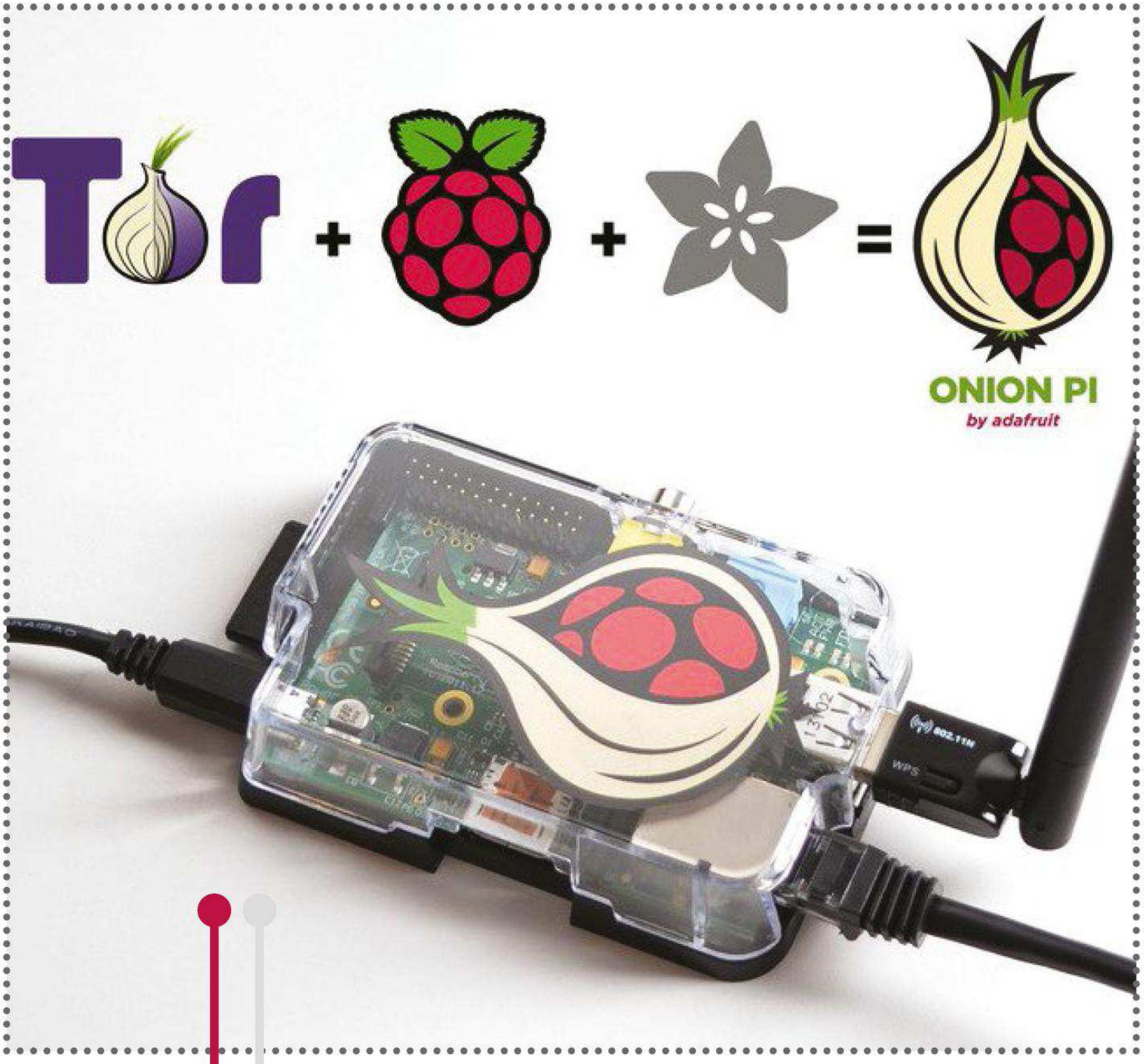
I notice you've had some useful feedback since your publication.

My initial version involved wire-wrapping the prongs of a USB power adaptor, so I could connect it directly to 120 VAC Romex. There simply wasn't space in a 3-gang electrical box to mount a power outlet, so things needed to be hard-wired, and a 4-gang box would have required a much larger bezel to cover it. I got a lot of feedback that this approach didn't feel safe, and one person suggested a way to use a C7 extension cord that would avoid the need to wire-wrap the prongs. I was thankful for the suggestion, bought those parts, and updated the design. I also created a version that would be powered using Power over Ethernet (POE), which eliminates the need for a high-voltage connection at all. My final project demonstrates both methods of powering the screen.



# Turn your Pi into a Tor proxy with Onion Pi

## Turn your Raspberry Pi into a wireless access point to access the anonymous Tor network



In this age of ubiquitous surveillance it's harder than ever to stop hackers, advertisers and shadowy government organisations from snooping on your browsing habits.

However if you choose to connect through Tor, your connection is encrypted and passed through a number of proxies through a process known as 'onion routing'. While this does slow down your connection, it also increases your privacy, making it extremely difficult to trace your actual current location.

Follow the steps in this tutorial to turn your Pi into a wireless AP (Access Point) named Onion_Pi. Any devices connecting to Onion_Pi will do so over the Tor network. When you're done, use a service like www.whatismyip.com to see that your location has changed.

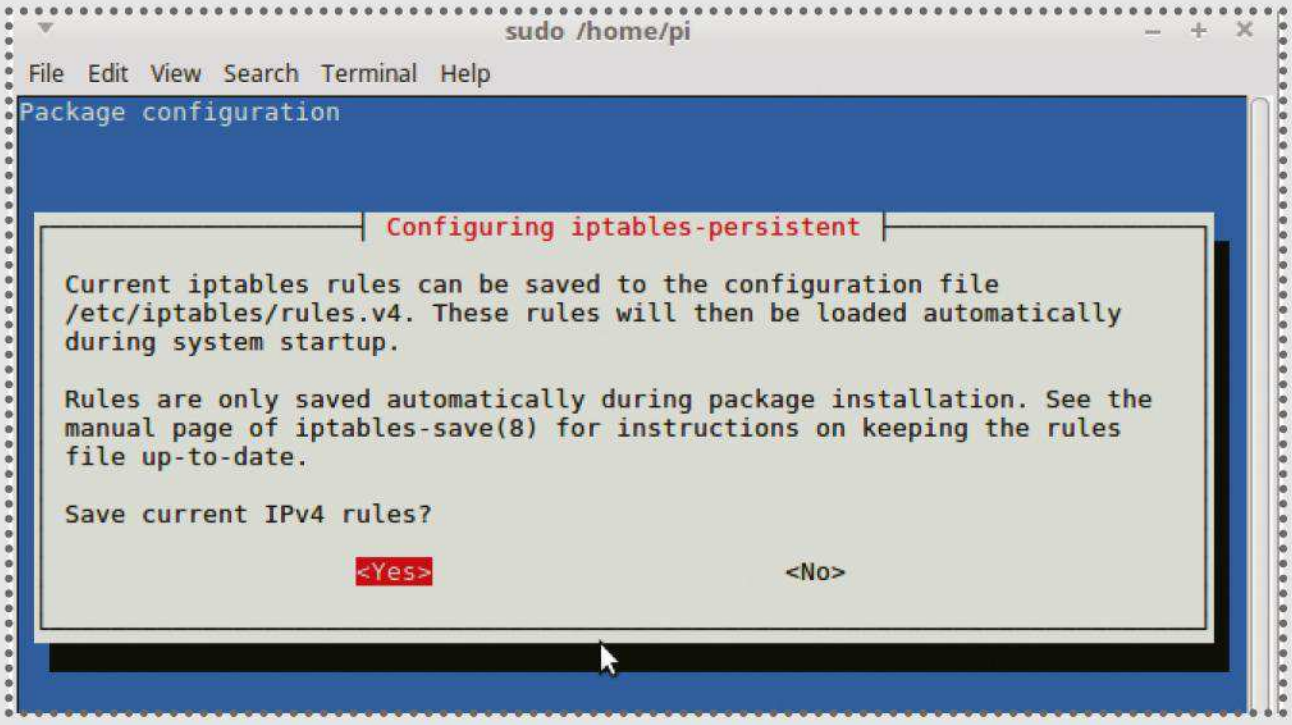
For more information about Tor visit https://www.torproject.org/about/overview.html.en.

## 01 Connect to Pi and check wireless is detected

Attach your Pi to your router via the Ethernet cable, then either open Terminal on the Pi or connect to it via SSH. Run the command sudo ifconfig -a. You should see the text wlan0 which shows that the wireless module is up and running.

## 02 Install essential software

Run the command sudo apt-get update then sudo apt-get install hostapd isc-dhcp-server tor iptables-persistent to install the necessary software. When you install iptables-persistent you'll be asked if you want to save the rules for your current configuration. Select 'Yes' both times.





## 03 Configure the DHCP server

Run sudo nano /etc/dhcp/dhcpd.conf. Find the two lines beginning 'option domain-name' and put a '#' at the start of each. Remove the '#' from the line '#authoritative'.

Scroll to the end and type:

```
subnet 192.168.42.0 netmask 255.255.255.0 {
  range 192.168.42.10 192.168.42.50;
  option broadcast-address 192.168.42.255;
  option routers 192.168.42.1;
  default-lease-time 600;
  max-lease-time 7200;
  option domain-name “local”;
  option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

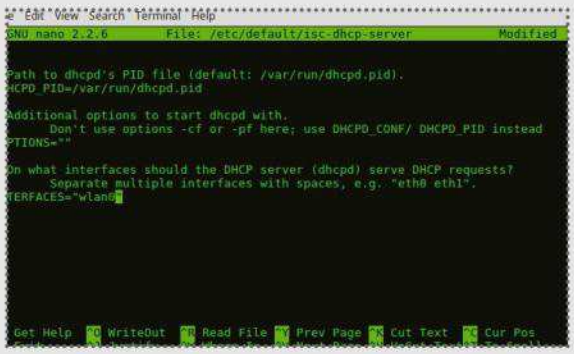
## 04 Edit interfaces

Run sudo nano /etc/default/isc-dhcp-server. Scroll down to the word INTERFACES="" and insert 'wlan0'. Press Ctrl

## How does Tor keep you safe?

The Tor network is a group of servers or 'relays' operated by volunteers. When you start tor on the Pi, it will build a circuit of encrypted connections through relays on the network. Each relay only knows the last relay a data packet came from and where it's going, meaning it's extremely difficult to trace the data back to you. Tor also changes the circuits it uses every few minutes to make it even harder to find your machine.

+ X, Y, then return to save and close.




Run the commands sudo update-rc.d hostapd enable and sudo update-rc.d isc-dhcp-server enable to make sure your changes start on boot.

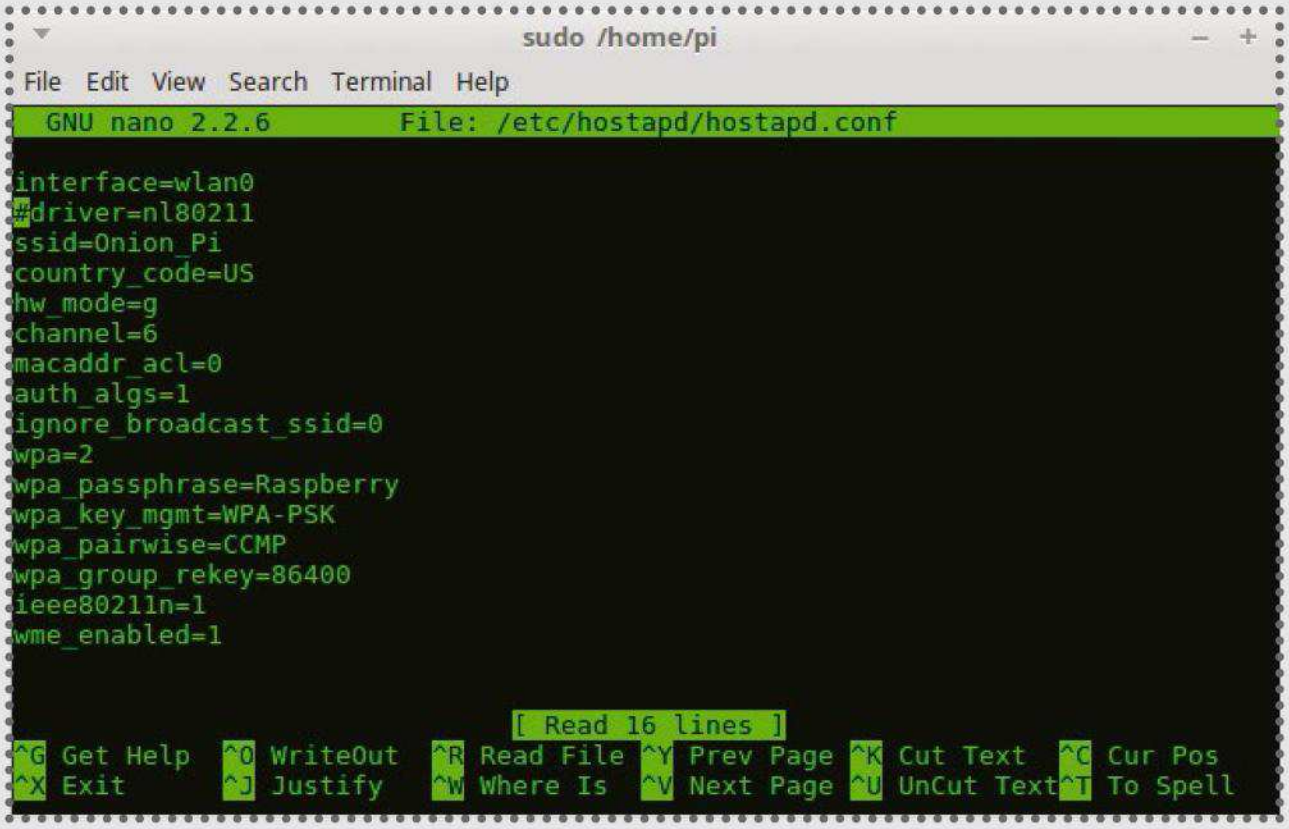
```
  GNU nano 2.2.6          File: /etc/network/interfaces                M

iface eth0 inet manual

allow-hotplug wlan0
    iface wlan0 inet static
      address 192.168.42.1
      netmask 255.255.255.0

#allow-hotplug wlan1
#iface wlan1 inet manual
#    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To
```

## 05 Set static IP

Run sudo nano /etc/network/interfaces. If you see the text 'auto wlan0' add a '#' at the start to comment it out. Find the line allow-hotplug wlan0 and delete the two lines below it. Replace them with these three lines:

```
iface wlan0 inet static
address 192.168.42.1
netmask 255.255.255.0
```

Run sudo ifconfig wlan0 192.168.42.1 to set your IP.



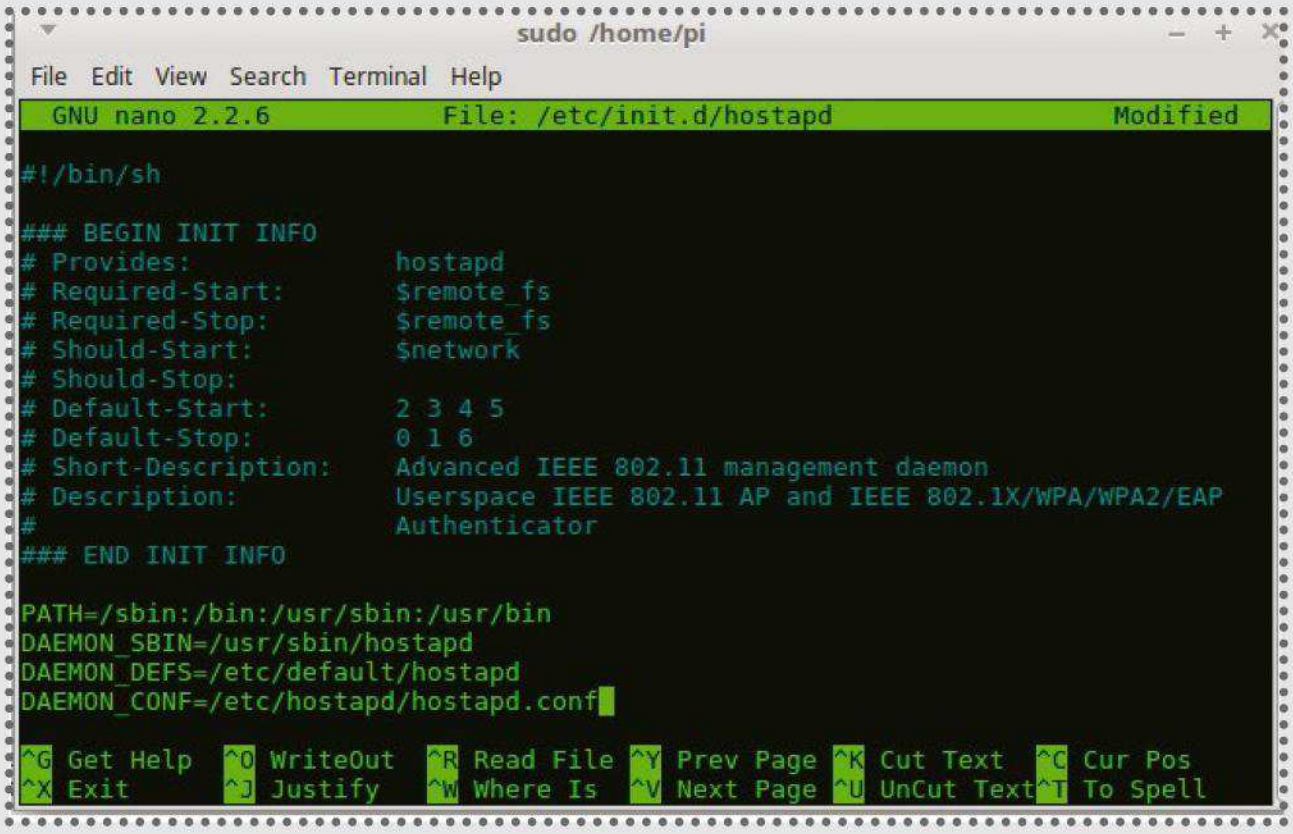

## 06 Configure the Access Point

Run sudo nano /etc/hostapd/hostapd.conf to create a blank file. Paste in the following:

```
interface=wlan0
driver=nl80211
ssid=Onion_Pi
country_code=US
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=Raspberry
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
wpa_group_rekey=86400
ieee80211n=1
wme_enabled=1
```

"Any devices connecting to Onion_Pi will do so over the Tor network."

Feel free to change the passphrase from 'Raspberry' to something more complex.




## 07 Apply Access Point configuration

Run sudo nano /etc/default/hostapd. Find the line:

```
#DAEMON_CONF=""
```

Edit it so it says:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Don't forget to remove the # in front to activate it, or the line won't work.

Repeat these same steps for hostapd with the command sudo nano /etc/init.d/hostapd again modifying the line #DAEMON_CONF="" so that it reads DAEMON_CONF="/etc/hostapd/hostapd.conf"

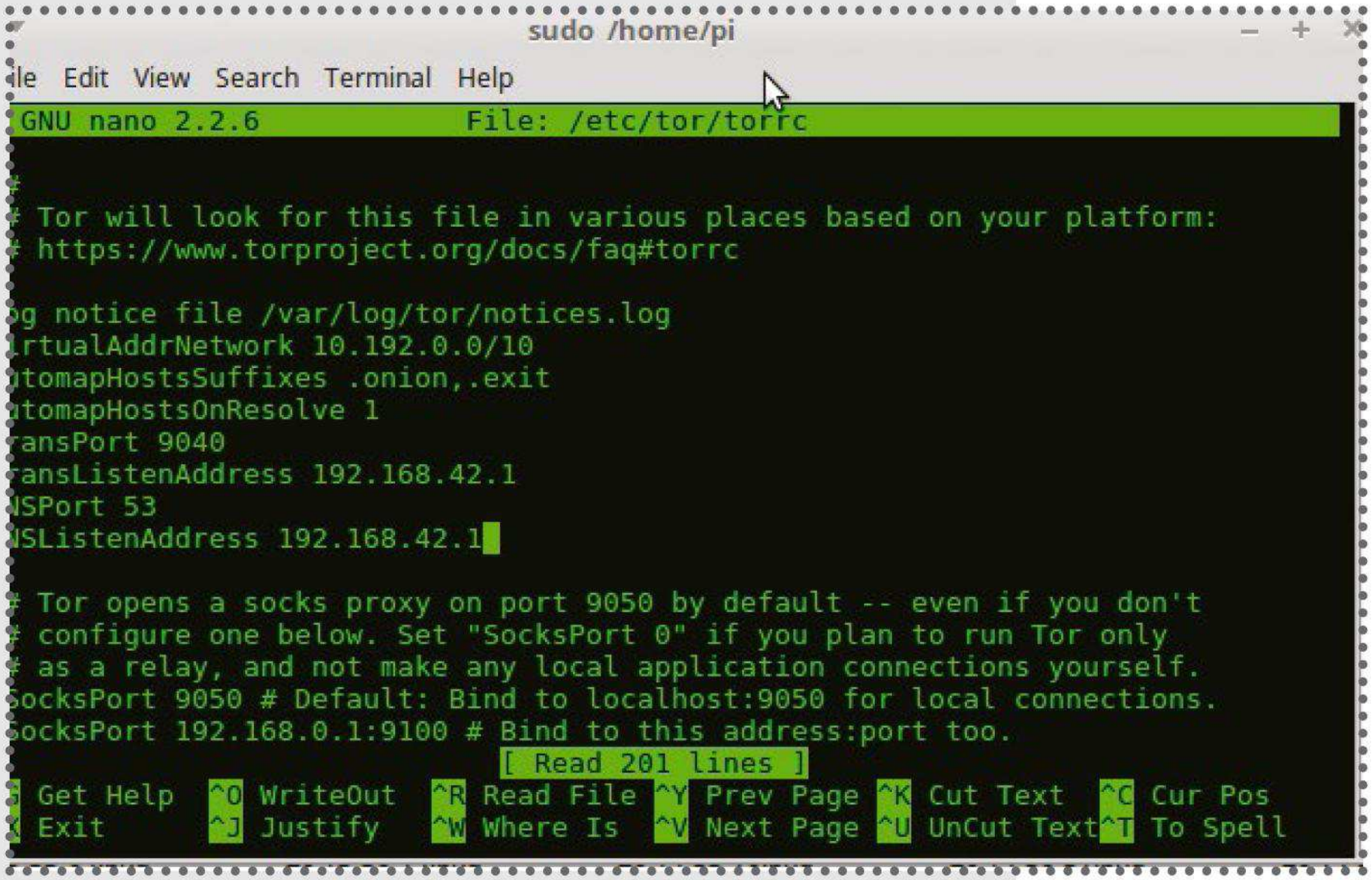
## Tor best practices

Don't use bit torrent: this places an unfair burden on volunteers running Tor relays. Bit torrent software also often sends out your real IP address.

Use HTTPS versions of websites: this encrypts your connection between your Tor exit node and your target website e.g https://gmail.com

Don't use browser plugins: this makes it easy to fingerprint your browser.

Don't open online documents: PDF and .doc files can contain code that might give away your real location.




## 08 Configure Tor

Run sudo nano /etc/tor/torrc to configure Tor. Find ## https://www.torproject.org/docs/faq#torrc and after it paste:

```
Log notice file /var/log/tor/notices.log
VirtualAddrNetwork 10.192.0.0/10
AutomapHostsSuffixes .onion,.exit
AutomapHostsOnResolve 1
TransPort 9040
TransListenAddress 192.168.42.1
DNSPort 53
DNSListenAddress 192.168.42.1
```

Save and exit. Run sudo update-rc.d tor enable to make

Tor start on boot.

## 09 Configure IP Tables

Run these commands to channel all traffic through Tor:

```
sudo iptables -t nat -A PREROUTING -i wlan0
  -p tcp --dport 22 -j REDIRECT --to-ports 22
sudo iptables -t nat -A PREROUTING -i wlan0
  -p udp --dport 53 -j REDIRECT --to-ports 53
sudo iptables -t nat -A PREROUTING -i wlan0
  -p tcp --syn -j REDIRECT --to-ports 9040
```

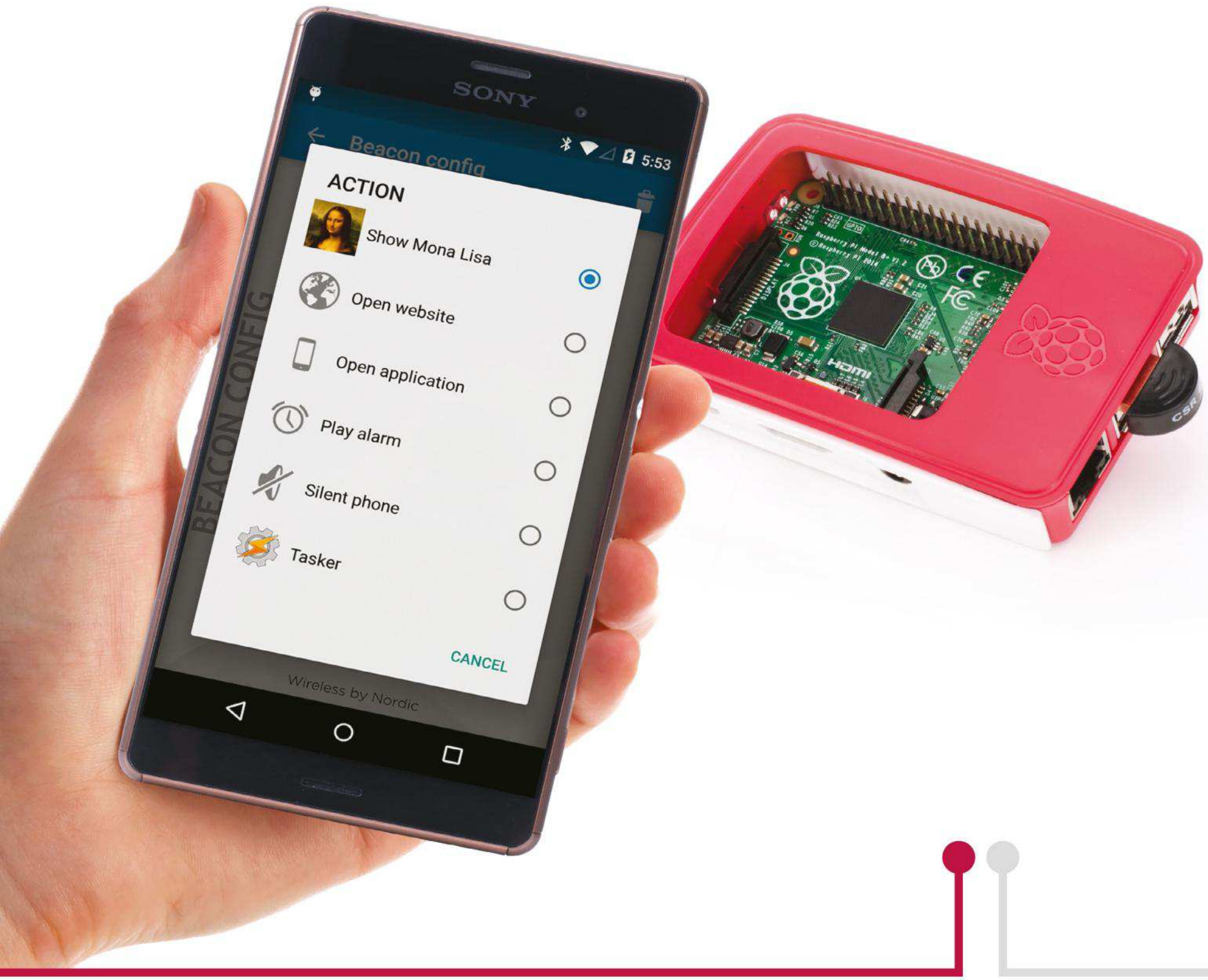
Next, make your changes permanent:

```
sudo sh -c “iptables-save >  /etc/iptables/
  rules.v4”
```

Reboot your Pi when done.

# Use your Raspberry Pi to find and track your phone

## Create a program that locates Bluetooth devices and responds to them



The Raspberry Pi model 3 saw the introduction of embedded Wi-Fi and Bluetooth capabilities. This now makes it even easier to interact with Bluetooth-enabled devices such as mobile phones, tablets and speakers. The Python programming language supports a range of libraries that enable you to interact, monitor and control various elements of a Bluetooth device. This tutorial combines the Pi's Bluetooth hardware with Python code to create three simple but useful programs. First, build a short program to search for Bluetooth-enabled devices and return the 12-part address of each device. Once you have obtained the addresses, you can then scan and find Bluetooth services that are available on that particular device. Finally, use the Bluetooth address and some conditions to check which devices are present in a building and in turn, which people are present or absent in a building, responding with a desired action – it's a kind of automated Bluetooth checking-in system.

**THE PROJECT ESSENTIALS**

**Raspberry Pi 3**

**Bluetooth USB dongle (if using an older Pi model)**

## 01 Using Bluetooth

Bluetooth is a wireless standard for exchanging data between devices over short distances of between one and ten metres. The current version, Bluetooth v5, was notified in June 2016 and has an increased range of over 200 metres. It will be released in 2017 and will probably be a staple of many IoT devices and applications. Bluetooth uses the standard IEEE 802.11, which is the same standard as Wi-Fi. They both have similarities such as setting up a connection, transferring and receiving files and streaming audio and media content. If you have a Pi 2 or lower, you can still create and use these

programs by using a Bluetooth USB dongle.

## 02 Install the required libraries

Although the Raspberry Pi OS image comes with a Bluetooth library for interfacing with devices, for this tutorial you will want to control the interface with Python code. Load up your Pi and open the LX Terminal window. Check for and update/upgrade the OS, typing in lines 1 and 2. Then install the Python development tools, line 3. Finally install two further Bluetooth development libraries. Once they have completed, restart your Pi by typing sudo halt.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python-pip python-dev
  ipython
sudo apt-get install bluetooth libbluetooth-
  dev
sudo pip install pybluez
```

## 03 Load Python

Load the LX Terminal and type sudo idle, this will open the Python editor with super-user privileges, which will give you access to the USB hardware via Python code. Start a new Window and import the first Bluetooth module, line 1. Then add a short message to inform the user that the program is searching for nearby devices.

```
import Bluetooth
print("Searching for device...")
```

## 04 Search for the names of the devices

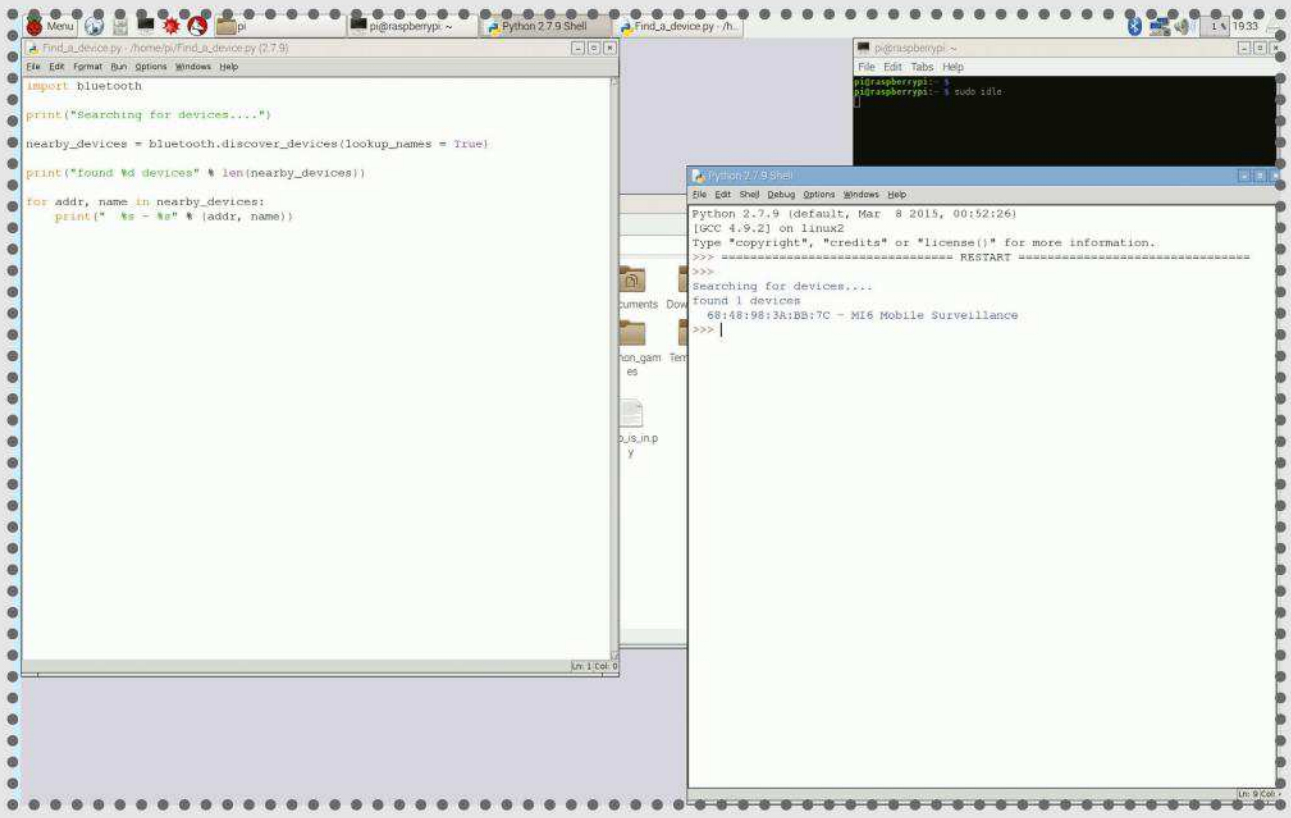
The next line of your program searches for the names of the Bluetooth enabled devices. Each device must have Bluetooth enabled and be discoverable in order to be found. On the next line down, create a variable called nearby_devices to store the names and use the code bluetooth.discover_devices to look up the names of the devices.

```
nearby_devices = bluetooth.discover_
devices(lookup_names = True)
```

## 05 Print the total number of devices found

Each of the names of any discoverable devices are now stored in a variable. Use the code len(nearby_devices) to return the number of items stored in the variable, this is the number of Bluetooth devices the program has found. Then print out the total number of devices. Add the following code after the previous line of your program.

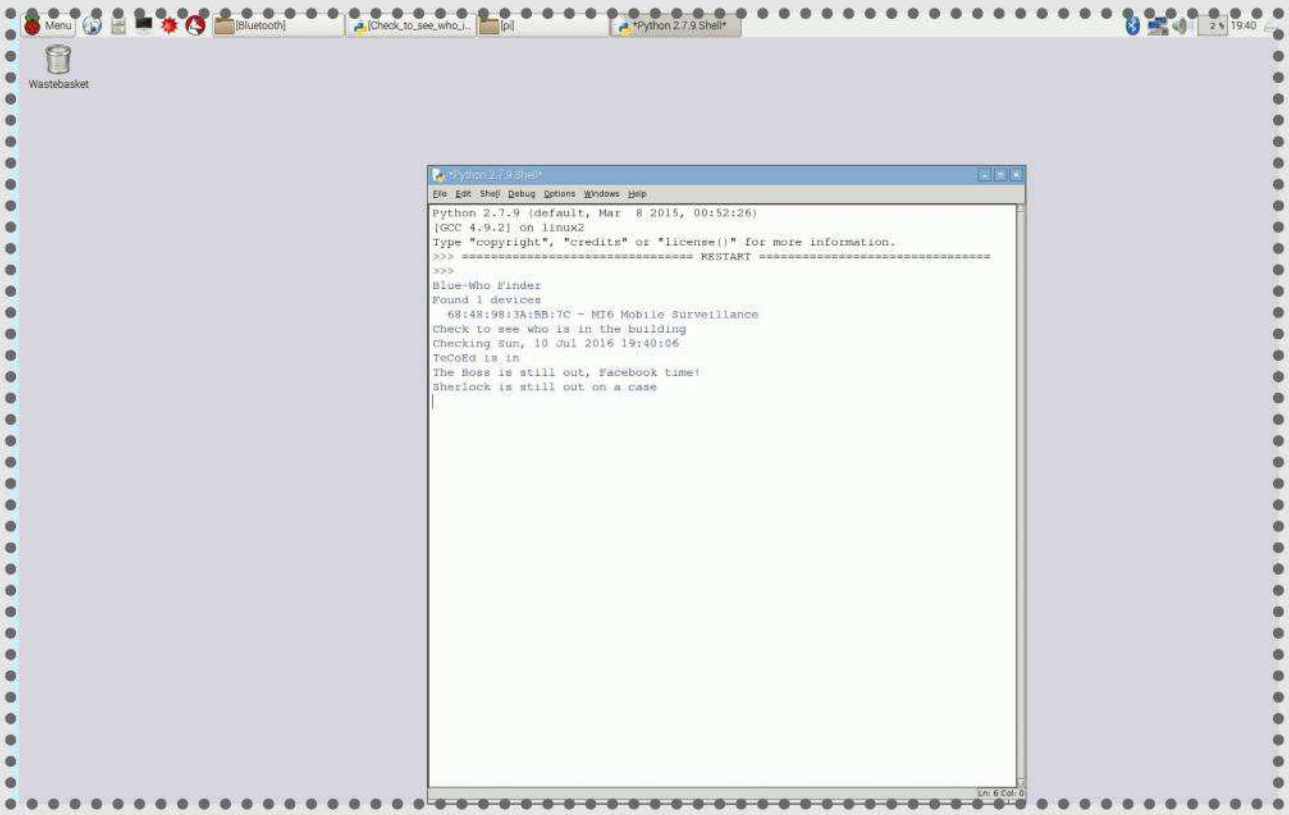
```
print("found %d devices" % len(nearby_
devices))
```




**Left** Once everything is correctly set up, you'll be greeted with this intro page

## 06 The Bluetooth address (BD_ADDR)

Each Bluetooth-enabled device has a Bluetooth address that is a combination of 12 alphanumeric characters; for example, 69:58:78:3A:CB:7F. The addresses are hexadecimal, which means they can contain numbers from 0 to 9 and letters from A to F. Most device




manufactures will include the address on a sticker attached to the hardware or within the user manual.

## 07 Print the name and address of the device

For each of the devices that the program has located, (each of the items in the list), print out the address of the device and also the name of the device. This information is used in the next section to find out what available services a device has and also to add an action when a device is found. Save the program and then run it, ensuring that the any devices to be found are in Discovery mode. You will be presented with a list of the devices that includes the name and address of each.

```
for addr, name in nearby_devices:
```

```
    print("  %s - %s" % (addr, name))
```

## 08 Find the available services on a Bluetooth device

Run the previous program and write down the Bluetooth address of the device. Start a new Python program and save it. Next open up a new Python window and start a new program. Input the two required libraries, lines 1 and 2.

```
#!/usr/bin/env python
import bluetooth
from bluetooth import *
```

## 09 Set up the address to find

On the next line down, enter the address of the device that you want to find the services on. Use the previous program or look at the sticker to obtain the address. Next, create a variable called 'device_address' to store the address. Use the following code and replace the example address with the Bluetooth address of your device.

```
device_address = "69:58:78:3A:CB:7F" # enter
address of device
```

## 10 Find a service

On the next line down, add the code to find the services that your device supports. Create a new variable called services, which will store a list of the services. Use the code, find_services, followed by the Bluetooth address of your enabled device to search through a list of available services and store each of them in the 'services' variable.

```
  services = find_service(address=device_
    address)
```

## 11 Print out each service

The last step of the program is to print out each of the services. These are stored in a list and therefore need to be printed out one line at a time. Create a loop using the code, for i in services, line 1. This loop will check for each of the individual items in the list. It will then print each of the items in the list, each Bluetooth service, line 2. Use the code '\n' to print each service onto a new line.

```
  for i in services:
    print i, '\n'
```

## 12 What are the services?

Save and run your program and you will be presented with a long list of services, especially if you are using a modern device. Using these services with Python is more complex and requires several other lines of code. However, you can potentially transfer and receive files, stream music and even shut the device down. There are more details and commentary on the Blueman Github page, https://github.com/blueman-project/blueman. Remember though that these tools are purely for personal use.

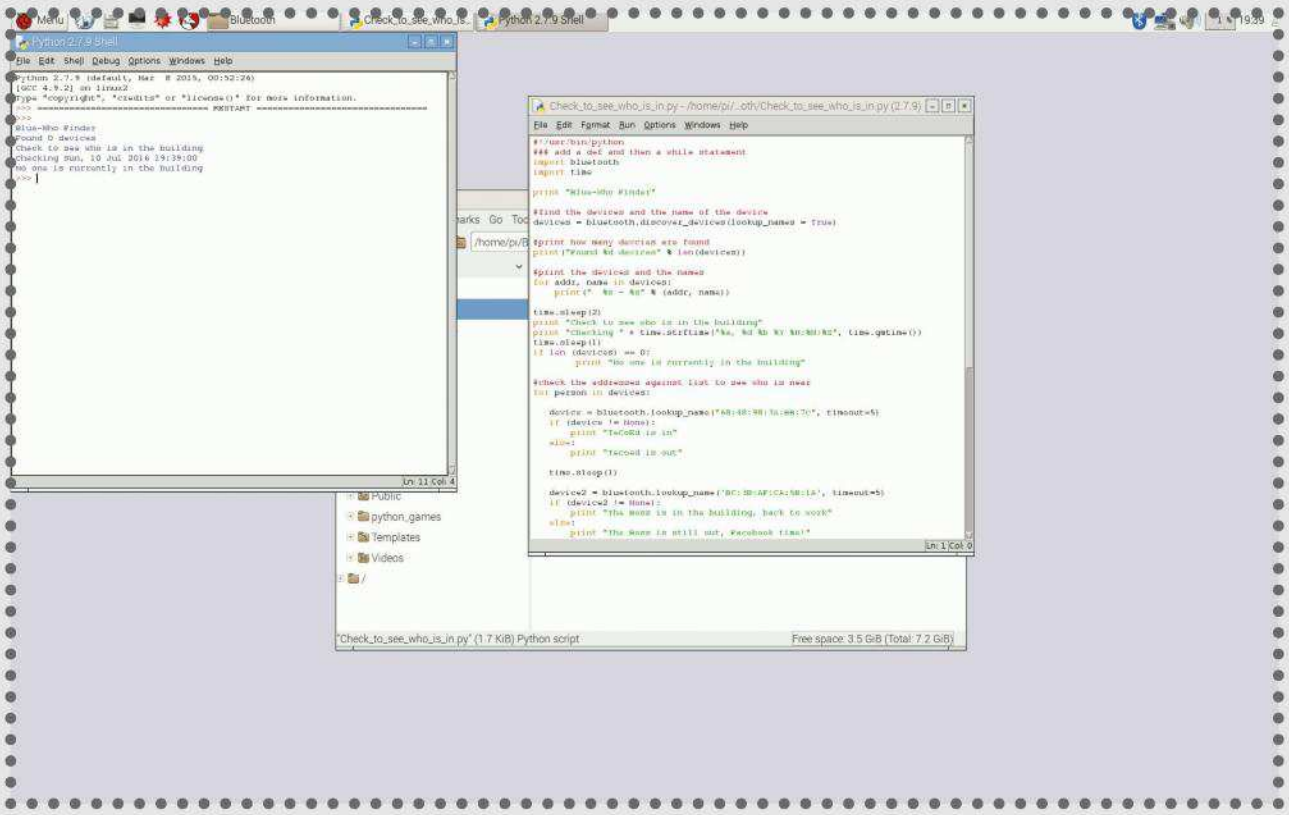
## 13 Find a device and a person

In Step 7 you used a short program to discover the Bluetooth-enabled device and check and return the address of the device. Now use the bluetooth.lookup_ name code line to search for a particular device and

## Is your dongle working?

If you are using a USB Bluetooth Dongle then you can check that it is working by plugging it in, then restarting your Raspberry Pi by typing the command sudo reboot. When reloaded, check that the Dongle has been recognised; load the LX Terminal window and type lsusb. This will list all the connected USB devices. You can also type hcitool dev to identify the dongle and return its USB address.

return whether it is found or not. If it is found then the device is present and if not, then we can assume the device is not. However, remember that the Bluetooth may be turned off. In your Python program add the line




to locate the device, replacing the example address with the address of the one that you want to locate.

```
device = bluetooth.lookup_
name("33:38:33:6A:BQ:7C", timeout=5)
```

## 14 Respond if the device is found

Once a device has been searched for, check to see if it is present and responds, line 1. Use an IF statement to see if the device is not found. This uses the symbol '!=', which means 'is not'. However, the code is checking if it is not 'None' – in other words the device is found. If it finds the named device then print out a message, line 2. If it does not find the device, line 3, then print out a message to notify the user, line 4. Add these lines of code underneath the previous line. Ensuring that your

Bluetooth is enabled, save and run the program to find your device.

```
if (device != None):
      print “TeCoEd is in”
   else:
      print “Tecoed is out”
```

## 15 Find a different device

To find other devices and respond with an action, simply use the same sequence of code but first create a different variable to store the response in. Add the code on the next line down and remember to de-indent it. Rename the variable, for example call it device_one and edit the address to match that of the second device.

```
Device_one = bluetooth.lookup_
  name(“44:67:73:6T:BR:7A”, timeout=5)
```

## 16 Another response, the next device is found

As before, check and respond, line 1, using an IF statement to see if the device is not found. Remember to use the new variable name, in this example, device_one. If it finds the named device then print out a message, line 2. If it does not find the device, line 3, then print out a message to notify the user, line 4. Add these lines of code underneath the previous line. Save and run the program to find the two particular devices within your location.

```
   if (device_one != None):
      print “Linux Laptop is in”
   else:
```

## Is your Bluetooth Dongle compatible?

If you have an older Raspberry Pi model 1, 2 or the Pi Zero then Bluetooth capability is not included. However, you can buy a USB Bluetooth dongle and attach it via one of the USB ports to add capability. Not all Bluetooth dongles are compatible so there is a developing list of the tried and tested ones available. Check here before you purchase one: http://elinux.org/RPi_USB_Bluetooth_adapters

```
print “Linux Laptop is out”
```

## 17 Add an alternative action

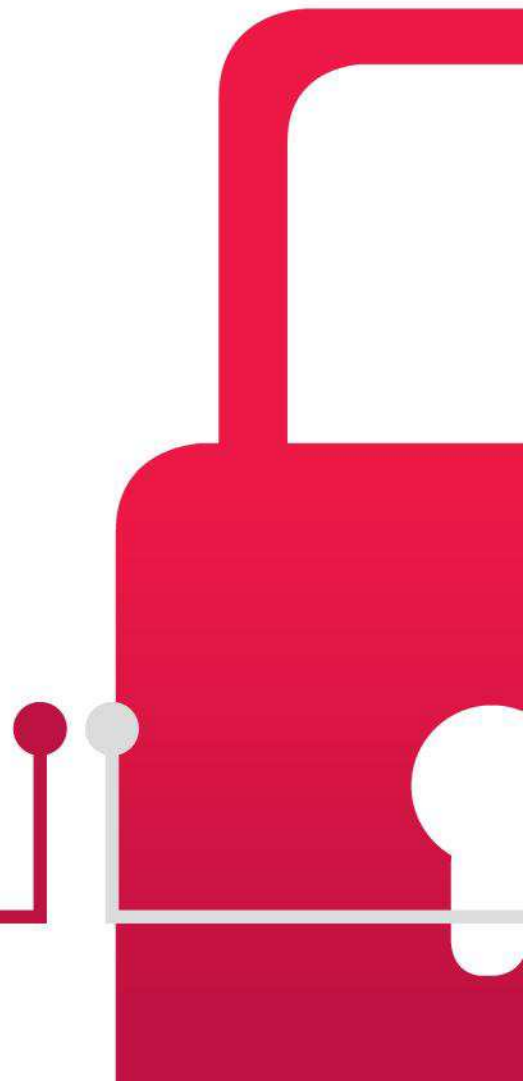
To customise the project further you could add your own action to the devices on being discovered. For example, use a strip of LEDs to flash each time a device enters the location and is detected. Or use individual LEDs for each individual device to indicate if the device is present or not. How about combining the program with an LCD screen as a message board for who is present and who is out? For more inspiration, check out **https://www.youtube.com/watch?v=qUZQv87GVdQ**

# Using your RDBMS with Python

## When you have a lot of data to work with, you will likely need to use a RDBMS – so this month, learn how to use one with Python

Last issue, we looked at how to use SQLite to use data without needing a full RDBMS (Relational DataBase Management System). This is fine when you have a limited amount of data, but at some point you'll need the extra performance. Options such as MySQL or Postgresql are available that focus on providing your data as efficiently as possible. We won't be looking at how to set up or manage the database; instead, we'll assume that there is already an existing database and focus on how to use it with Python. Also, we'll be using MySQL as the example RDBMS; the concepts are very similar from one database to another with only the syntax changing much. To install the Python module for Debian-based distributions, such as Raspbian, use sudo apt-get install python-mysql.connector. If you are using Python 3.x, you can replace python-mysql.connector with python3-mysql.connector.

Once the Python module is installed, import it into your program with import mysql.connector. The first step is to connect to the MySQL server. The basic form



is:

```
my_conn = mysql.connector.
  connect(user='username',
  password='password',
  host='127.0.0.1', port='3306',
  database='mydb')
```

In this example, the MySQL service is running on the local machine, hence the host being set to 127.0.0.1. If it is running on another machine, you can set the host parameter to the relevant IP address or hostname. If it's listening on the default port, 3306, you can leave it off the parameter list. In previous articles, we haven't worried about exception handling, but when it comes to connecting to database, we should look at how to manage possible connection errors:

```
from mysql.connector import errorcode
try:
  my_conn = mysql.connector.
  connect(user='username', database='test1')
except mysql.connector.Error as err:
  if err.errno == errorcode.ER_ACCESS_DENIED_
  ERROR:
    print("Something is wrong with your user
  name or password")
  elif err.errno == errorcode.ER_BAD_DB_
  ERROR:
    print("Database does not exist")
else:
  print(err)
else:
```

```
my_conn.close()
```

This all assumes that the database already exists on the MySQL server. If it doesn't, you can leave the database parameter out of your connect call, and create the database after connecting to the server. The following code would create a new test1 database:

```
DB_NAME = 'test1'
my_conn = mysql.connector.
  connect(user='username')
my_cursor = my_conn.cursor()
my_cursor.execute("CREATE DATABASE {} DEFAULT
  CHARACTER SET 'utf8'".format(DB_NAME))
my_conn.database = DB_NAME
```

This way, you can have your program bootstrap the entire data storage step, assuming that the username you are using has the privileges needed to create a new database.

Continuing the setup, you may need to create tables to store your data before doing any work with it. Just as with creating a database, you will need to have a cursor that can execute SQL statements. The following code will create a table to store names and phone numbers:

```
table_stmt = "CREATE TABLE 'phones' ('name'
  varchar(50) NOT NULL, 'number' int(9) NOT
  NULL) ENGINE=InnoDB"
my_cursor.execute(table_stmt)
```

As you can see, we are just handing in SQL

## Why Python?

It's the official language of the Raspberry Pi. Read the docs at python.org/doc

statements to be processed by the MySQL server. These types of statements are called DDL (Data Definition Language) statements and you can send in pretty much anything that the MySQL server will understand.

Once your database has been created and properly structured, you need to load data in order to start using it. If you have large amounts of data, you will likely want to bulk-load it directly using the utilities that come with MySQL. If you are loading data as it is being collected within your program, use:

```
  add_phone = "INSERT INTO phones (name,
number) VALUES (%s, %s)"
phone_data = ('Joey Bernard', 5551234567)
my_cursor.execute(add_phone, phone_data)
```

As you can see, we separated out the insertion statement from the data being inserted. This way, you can easily reuse the add statement. Also, since the data is separated out, you can more easily do pre-processing to ensure that the incoming data is sanitised. One of the key structures of a RDBMS is the relational part. This means that you may need the row ID of the most recent insertion to use as a key linking it to some other entry in another table. This would look like

```
row_id = my_cursor.lastrowid.
```

By default, connections to the database have autocommit turned off. This means that everything you do is handled through transactions. To ensure that the data change you just made is pushed to the

database, you need to commit the transaction with my_cursor.commit().

This will commit everything that has happened since the last commit call. This means that you can also rollback changes using the rollback() method of the cursor object. Again, this applies to everything that has happened since the last commit.

Once you have a database that's fully loaded with data, how do you pull it back out in order to work with it? You can hand in a SELECT statement, using the execute() method of the cursor object. As with the INSERT statement above, you can separate the statement from any search parameters that you want to use to constrain your query. For example, the following code will pull up all of the data in the test1 table:

```
my_cursor.execute("SELECT * FROM test1")
```

There are two ways to pull out the results from this query. If you want to pull out one of them, you can use the fetchone() method of the cursor object. This will give you a tuple containing the next row in the list of rows returned by your query. There are also fetchmany() and fetchall() methods that allow you to grab larger chunks of returned data. If you wanted to step each returned row and do something with each one, you can use:

```
 for (name,number) in my_cursor:
    print("Name: {}, Phone number: {}".
    format(name, number))
```

This works because the cursor object can be used as an iterator.

"We should manage connection errors"

As users work with your program, you will need to alter data stored in your MYSQL database. If you need to update stored information, you can use the UPDATE SQL statement. The code below will update my phone number:

```
  my_cursor.execute("UPDATE test1 SET
number=5559876543 WHERE name='Joey
Bernard'")
```

If you find that you need to clean up old data, you can remove it with:

```
  my_cursor.execute("DELETE FROM test1
WHERE name='Joey Bernard'")
```

When your data collection gets large enough, you may want to take advantage of the strengths of an RDBMS by creating and using stored procedures within the MySQL database. We'll assume that you have already created a stored procedure within the database, named my_func. You can then use the callproc() method of the cursor object:

```
  my_cursor.callproc('my_func')
for result in my_cursor.stored_results():
  print(result.fetchall())
```

You need to use the stored_results() method to pull each result out and then use its fetchall() method to get the actual returned data. When you are done, don't forget to clean up after yourself with:

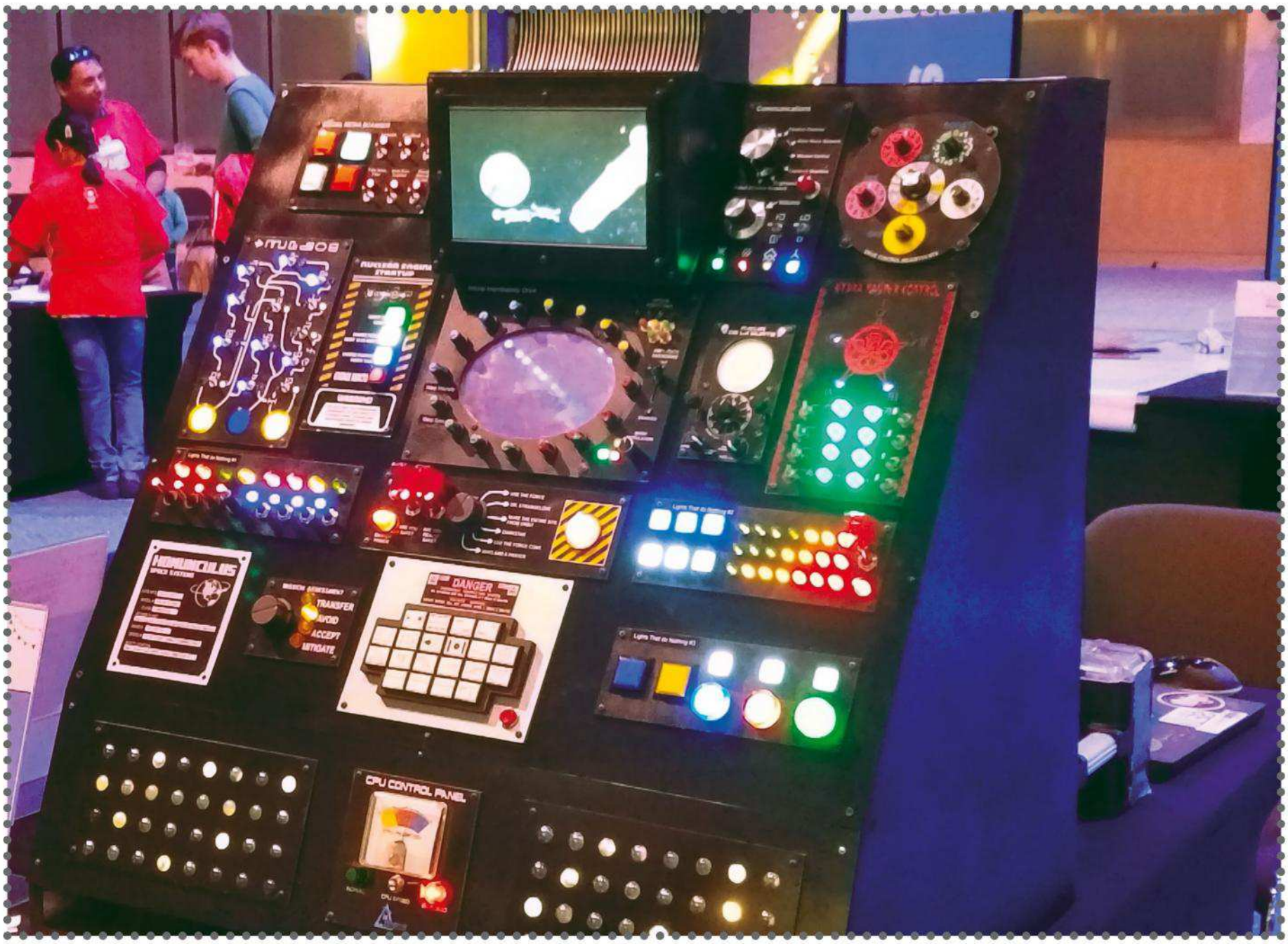
```
  my_cursor.close()
my_conn.close()
```

And now you are ready to handle even larger amounts of data!

# Next issue

**Get inspired** **Expert advice** **Easy-to-follow guides**

## "Sci-fi Pi project revealed"




Get this issue's source code at:

**www.linuxuser.co.uk/raspicode**